



SOFE 3980U Software Quality

Assignment 2

Name: Daniyal Khan (100750029)

Due Date: March 24th ,2022

Problem A Github Link: (<https://github.com/100750029/Software-Quality-Assignment1>)

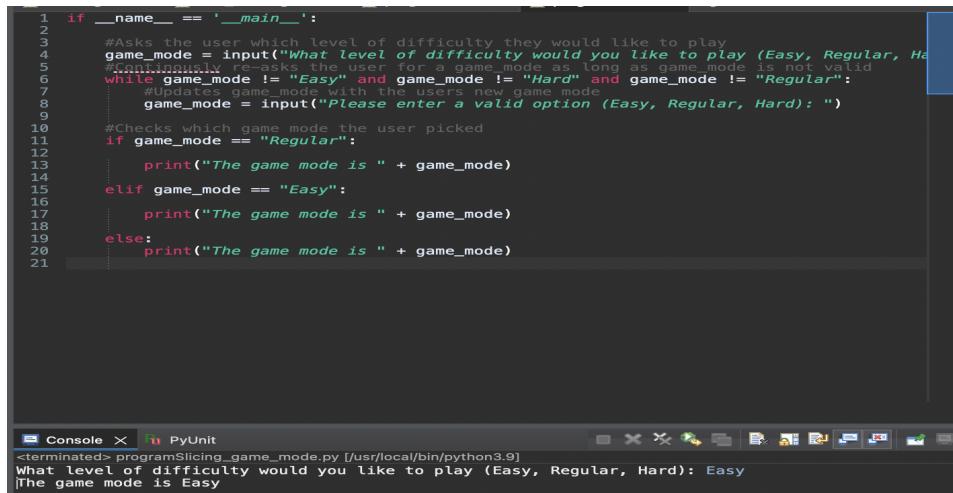
Static Analysis Techniques:

An example of a static analysis technique is program slicing. Program slicing is focusing on a specific selection of variables within a program. A program slice refers to the sections of the program that are relevant to the subset of variables. There are four types of program slicing:

- Forward:
 - A forward slice across a program for a given statement X contains all statements that are affected by X.
- Backward:
 - A backward slice of a program for a given statement X contains all statements that affect whether control reaches X as well as all statements that affect the value of variables that occur in X.
- Static:
 - A static program slice is calculated symbolically, that is, without taking specific data values into account.
- Dynamic:
 - A dynamic program slice is created using specific data values.

For my program, I chose to do a mix of the following techniques listed above.

Program Slicing:



The screenshot shows a terminal window with Python code and its execution output. The code is a script named `programSlicing_game_mode.py` that asks the user for a game mode (Easy, Regular, Hard) and prints it back. The terminal shows the code being run and the user input "Easy".

```
1 if __name__ == '__main__':
2     #Asks the user which level of difficulty they would like to play
3     game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
4     .....
5     #Asks the user to enter a valid game mode as long as game_mode is not valid
6     while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
7         game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")
8
9     #Checks which game mode the user picked
10    if game_mode == "Regular":
11        print("The game mode is " + game_mode)
12    elif game_mode == "Easy":
13        print("The game mode is " + game_mode)
14    else:
15        print("The game mode is " + game_mode)
16
17
18
19
20
21
```

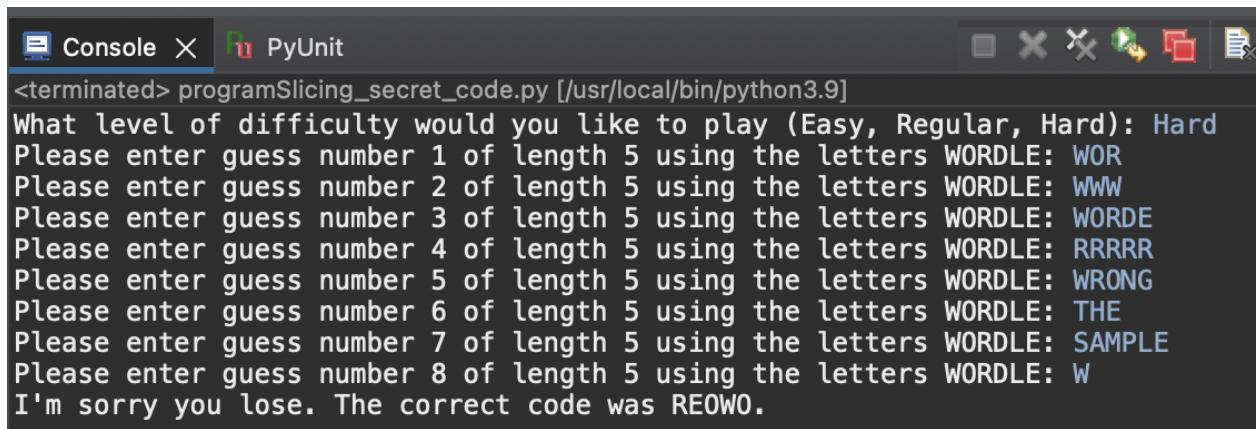
```
Console > PyUnit
<terminated> programSlicing_game_mode.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Easy
The game mode is Easy
```

For the first section, I applied program slicing to a variable called **game_mode**. The variable was only used in the main function and is used to select the difficulty of the game. I replaced the variables, SIZE and TRIES, with a print statement as they set the length of the word and number of attempts you get, and as such aren't necessary. The print statement outputs the difficulty you've selected.

```

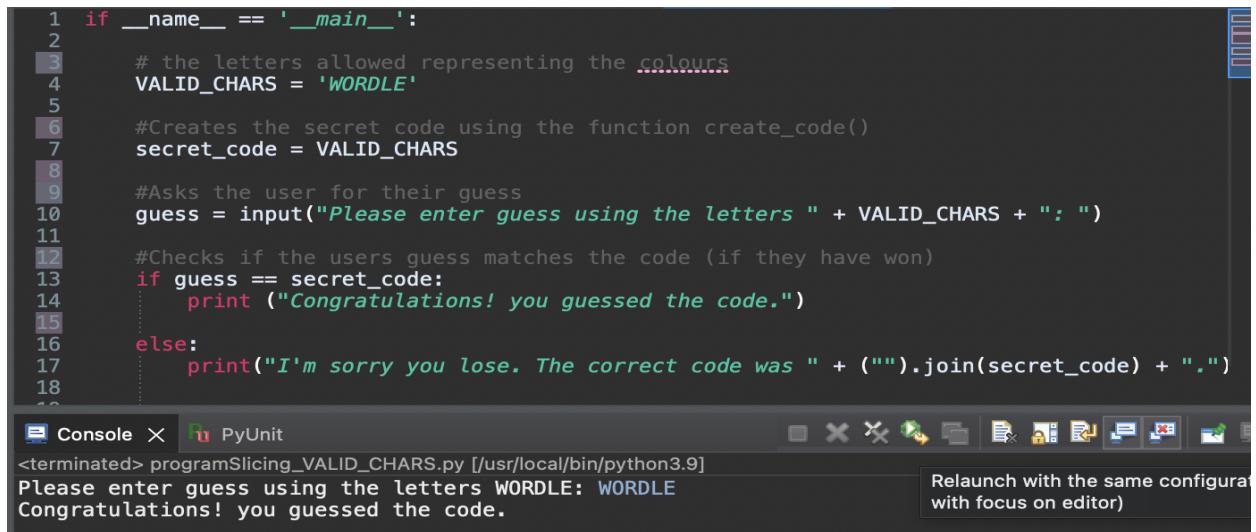
3● def create_code(characters, length):
4
5●     ...
6     Returns a list the length of length (int) with any
7     random single characters from characters (str).
8
9     >>> create_code("Danival", 4)
0     ['D', 'i', 'y', 'l']
1     >>> create_code("Hello", 3)
2     ['H', 'H', 'H']
3     ...
4
5     #Creates the list that will store the code
6     code = []
7     #Adds a random letter from characters to code length amount of times
8     for i in range(length):
9         code.append(random.choice(characters))
0
1     return code
2
3 if __name__ == '__main__':
4
5     #Asks the user which level of difficulty they would like to play
6     game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
7     #Continously re-asks the user for a game_mode as long as game mode is not valid
8     while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
9         #Updates game_mode with the users new game mode
0         game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")
1
2     #Checks which game mode the user picked
3     if game_mode == "Regular":
4         # the number of pegs in the answer for the game mode "Regular"
5         SIZE = 4
6         # the number of guesses the user gets for the game mode "Regular"
7         TRIES = 10
8     elif game_mode == "Easy":
9         # the number of pegs in the answer for the game mode "Easy"
0         SIZE = 3
1         # the number of guesses the user gets for the game mode "Easy"
2         TRIES = 12
3     else:
4         # the number of pegs in the answer for the game mode "Hard"
5         SIZE = 5
6         # the number of guesses the user gets for the game mode "Hard"
7         TRIES = 8
8
9●     # the letters allowed representing the colours
0     # green, red, blue, yellow, orange, purple
1     VALID_CHARS = 'WORDLE'
2
3     #Creates the secret code using the function create_code()
4     secret_code = create_code(VALID_CHARS, SIZE)
5
6     #Runs as many times as the users number of tries
7     for i in range(TRIES):
8         #Asks the user for their guess
9         guess = input("Please enter guess number " + str(i + 1) + " of length " + str(SIZE) + " using the letters " + VALID_CHARS + ": ")
0         #Converts the users guess into a list so that it can be compared to secret_code
1         guess = list(guess)
2         #Checks if the users guess matches the code (if they have won)
3         if guess == secret_code:
4             print ("Congratulations! It took you " + str(i + 1) + " guesses to find the code.")
5             #Breaks the for loop so that the user is not asked to guess again (the game has ended)
6             break
7
8●     #The secret_code will be given if the user looses because they have no tries left
9     # (the for loop finished and the user was unable to guess the right code)
0     if guess != secret_code:
1         print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")

```



```
<terminated> programSlicing_secret_code.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Hard
Please enter guess number 1 of length 5 using the letters WORDLE: WOR
Please enter guess number 2 of length 5 using the letters WORDLE: WWW
Please enter guess number 3 of length 5 using the letters WORDLE: WORDE
Please enter guess number 4 of length 5 using the letters WORDLE: RRRRR
Please enter guess number 5 of length 5 using the letters WORDLE: WRONG
Please enter guess number 6 of length 5 using the letters WORDLE: THE
Please enter guess number 7 of length 5 using the letters WORDLE: SAMPLE
Please enter guess number 8 of length 5 using the letters WORDLE: W
I'm sorry you lose. The correct code was REOWO.
```

Next I chose to do program slicing for the variable `secrete_code`. This variable involves the `create_code` function so we need to include it in our program. When I run the program it prompts me for the difficulty and then asks me to enter a guess. It doesn't give me any hints or clues because those variables aren't reliant on the `secrete_code` variable. It also doesn't check if the user guess is valid and it outputs the correct answer after you've reached the maximum number of tries.



```
1 if __name__ == '__main__':
2
3     # the letters allowed representing the colours
4     VALID_CHARS = 'WORDLE'
5
6     #Creates the secret code using the function create_code()
7     secret_code = VALID_CHARS
8
9     #Asks the user for their guess
10    guess = input("Please enter guess using the letters " + VALID_CHARS + ": ")
11
12    #Checks if the users guess matches the code (if they have won)
13    if guess == secret_code:
14        print ("Congratulations! you guessed the code.")
15    else:
16        print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")
17
18
<terminated> programSlicing_VALID_CHARS.py [/usr/local/bin/python3.9]
Please enter guess using the letters WORDLE: WORDLE
Congratulations! you guessed the code.
```

For the variable `VALID_CHARS`, I chose to do dynamic program slicing. I made it so the `secret_code` was already set to "WORDLE" which meant we didn't need any clues or hints. It doesn't check if the guess is valid either. Also, the program didn't need to create a code for the user to guess. The user just needs to input the guess and then the program will output if you got it correct or not.

```

1 def valid(user_guess, valid_characters, guess_size):
2     """
3     Returns True if all the characters in user_guess (list) are in
4     valid_characters (str) and the length of user_guess is equal
5     to guess_size (int).
6
7     >>> valid(['H', 'E', 'L', 'L', 'O'], "HELOYT", 5)
8     True
9     >>> valid(['S'], "S", 3)
10    False
11    >>> valid(['G', 'R', 'Y', 'O'], "GRBYOP", 4)
12    True
13
14    """
15
16    #Checks if the length of user_guess is equal to guess_size
17    if len(user_guess) == guess_size:
18        #Goes through each character in user_guess
19        for character in user_guess:
20            #Returns False if the character is not found in valid_characters
21            if character not in valid_characters:
22                return False
23
24    else:
25        return False
26
27    return True
28
29 if __name__ == '__main__':
30
31     #Asks the user which level of difficulty they would like to play
32     game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
33     #continually re-asks the user for a game_mode as long as game_mode is not valid
34     while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
35         #Updates game_mode with the users new game mode
36         game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")
37
38     #Checks which game mode the user picked
39     if game_mode == "Regular":
40         # the number of pegs in the answer for the game mode "Regular"
41         SIZE = 4
42         # the number of guesses the user gets for the game mode "Regular"
43         TRIES = 10
44     elif game_mode == "Easy":
45         # the number of pegs in the answer for the game mode "Easy"
46         SIZE = 3
47         # the number of guesses the user gets for the game mode "Easy"
48         TRIES = 12
49     else:
50         # the number of pegs in the answer for the game mode "Hard"
51         SIZE = 5
52         # the number of guesses the user gets for the game mode "Hard"
53         TRIES = 8
54
55     # the letters allowed representing the colours
56     # green, red, blue, yellow, orange, purple
57     VALID_CHARS = 'WORDLE'
58
59     #Runs as many times as the users number of tries
60     for i in range(TRIES):
61         #Asks the user for their guess
62         guess = input("Please enter guess number " + str(i + 1) + " of length " + str(SIZE) + " using the letters " + VALID_CHARS + "; ")
63         #Converts the users guess into a list so that it can be compared to secret_code
64         guess = list(guess)
65         #continually asks the user for a valid guess as long as their guess is not valid
66         while not valid(guess, VALID_CHARS, SIZE):
67             #Updates guess to store the users new guess
68             guess = input("Please re-enter a valid guess of length " + str(SIZE) + " using the letters " + VALID_CHARS + "; ")
69             #Converts the users guess into a list so that it can be compared to secret_code
70             guess = list(guess)

```

```

Console X PyUnit
programSlicing_valid.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Hard
Please enter guess number 1 of length 5 using the letters WORDLE: THE
Please re-enter a valid guess of length 5 using the letters WORDLE: WOR
Please re-enter a valid guess of length 5 using the letters WORDLE: WORD
Please re-enter a valid guess of length 5 using the letters WORDLE: WORDL
Please enter guess number 2 of length 5 using the letters WORDLE: WORDLE
Please re-enter a valid guess of length 5 using the letters WORDLE:

```

Next I did static program slicing for the function **valid**. This function includes the objects `user_guess`, `valid_characters`, and `guess_size`. This program checks if the user's guess is valid based on the `SIZE` and `VALID_CHARS`. The function asks the users for a difficulty then the user must input a guess. The function checks if the guess is valid based on the constraints.

```

1 if __name__ == '__main__':
2
3     #Asks the user which level of difficulty they would like to play
4     game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
5     #Continously re-asks the user for a game_mode as long as game_mode is not valid
6     while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
7         #Updates game_mode with the users new game mode
8         game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")
9
10    #Checks which game mode the user picked
11    if game_mode == "Regular":
12        # the number of guesses the user gets for the game mode "Regular"
13        TRIES = 10
14    elif game_mode == "Easy":
15        # the number of guesses the user gets for the game mode "Easy"
16        TRIES = 12
17    else:
18        # the number of guesses the user gets for the game mode "Hard"
19        TRIES = 8
20
21    # the letters allowed representing the colours
22    # green, red, blue, yellow, orange, purple
23    VALID_CHARS = 'WORDLE'
24
25    #Creates the secret code using the function create_code()
26    secret_code = VALID_CHARS
27
28    #Runs as many times as the users number of tries
29    for i in range(TRIES):
30        #Asks the user for their guess
31        guess = input("Please enter guess number " + str(i + 1) + " using the letters " + VALID_CHARS + ": ")
32        #Converts the users guess into a list so that it can be compared to secret_code
33        guess = list(guess)
34        if guess == secret_code:
35            print ("Congratulations! It took you " + str(i + 1) + " guesses to find the code.")
36            #Breaks the for loop so that the user is not asked to guess again (the game has ended)
37            break
38
39    #The secret_code will be given if the user loses because they have no tries left
40    #(the for loop finished and the user was unable to guess the right code)
41    if guess != secret_code:
42        print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")
43

```

```

<terminated> programSlicing_TRIES.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Hard
Please enter guess number 1 using the letters WORDLE: W
Please enter guess number 2 using the letters WORDLE: WEW
Please enter guess number 3 using the letters WORDLE: THR
Please enter guess number 4 using the letters WORDLE: WORD
Please enter guess number 5 using the letters WORDLE: WSED
Please enter guess number 6 using the letters WORDLE: LPK
Please enter guess number 7 using the letters WORDLE: LIJY
Please enter guess number 8 using the letters WORDLE: REDF
I'm sorry you lose. The correct code was WORDLE.

```

Finally I did static program slicing for a variable called **TRIES**. This variable assigns the user a number of attempts based on the difficulty they've selected. I manually created a `secret_code` because we didn't need the `create_code` function. Also we didn't need to include the SIZE of the word because we are only checking if the program gives you the correct number of tries. The program doesn't check if the guess you've entered is valid and outputs the correct answer after you've reached the maximum number of tries.

Problem B

Dynamic Analysis:

Instrumentation adds measurement probes to the code so that it may be observed while it runs. It is possible to accomplish this on numerous levels. There are various ways for various levels. Each approach has distinct overheads and levels of accuracy. To evaluate the run time of functions, I created a function called elapsed_time:

```
1❶ import time
2 import datetime
3
4
5❷ def elapsed_time(func):
6
7❸     def inside_function(*args):
8
9         start_time = time.time()
10        print(func.__name__, 'start time:', datetime.datetime.utcnow().strftime('%H:%M:%S.%f'))
11        remove_fully_correct_function = func(*args)
12        end_time = time.time()
13        print(func.__name__, 'end time:', datetime.datetime.utcnow().strftime('%H:%M:%S.%f'))
14        time_elapsed = datetime.datetime.utcnow().strftime('%H:%M:%S.%f')
15        print(func.__name__, 'Elapsed time:', time_elapsed)
16        return remove_fully_correct_function
17
18    return inside_function
19
```

This function uses the import time and datetime to get the elapsed time of a function. It gets the elapsed time by subtracting the end time by the start time. It then displays the time by month and date.

Instrumentation:

```
1❶ import time
2 import datetime
3 import random
4 |
5
6❷ def elapsed_time(func):
7
8❸     def inside_fuction(*args):
9
10        start_time = time.time()
11        print(func.__name__, 'start time:', datetime.datetime.utcnow().fromtimestamp(start_time))
12        create_code_function = func(*args)
13        end_time = time.time()
14        print(func.__name__, 'end time:', datetime.datetime.utcnow().fromtimestamp(end_time))
15        time_elapsed = datetime.datetime.utcnow().fromtimestamp(end_time-start_time)
16        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
17        return create_code_function
18
19    return inside_fuction
20
21 @elapsed_time
22❹ def create_code(characters, length):
23
24❺     ...
25     Returns a list the length of length (int) with any
26     random single characters from characters (str).
27
28     >>> create_code("DANIYAL", 4)
29     ['D', 'i', 'y', 'l']
30     >>> create_code("Hello", 3)
31     ['H', 'H', 'H']
32     ...
33
34     #Creates the list that will store the code
35     code = []
36     #Adds a random letter from characters to code length amount of times
37     for i in range(length):
38         code.append(random.choice(characters))
39
40     return code
41
42     create_code_function = create_code(['D', 'A', 'N', 'I', 'Y', 'A', 'L'], 4)
43     print("The create_code function will output: " + str(create_code_function))
44
```

```
Console × PyUnit
<terminated> dynamicAnalysis_create_code.py [/usr/local/bin/python3.9]
create_code start time: 2022-03-24 02:43:50.506638
create_code end time: 2022-03-24 02:43:50.506865
create_code Elapsed time: 00:00:00.000227
The create_code function will output: ['N', 'D', 'D', 'L']
```

The `create_code` function returns a list the length of `length` (int) with any random single characters from `characters` (str). This code is created using the characters “DANIYAL” with the length of 4. The code that is created is “NDDL” and the function runs with an elapsed time of 227 micro seconds.

```

1● import time
2 import datetime
3
4● def elapsed_time(func):
5
6●     def inside_function(*args):
7
8         start_time = time.time()
9         print(func.__name__, 'start time:', datetime.datetime.utcnow().timestamp())
10        remove_fully_correct_function = func(*args)
11        end_time = time.time()
12        print(func.__name__, 'end time:', datetime.datetime.utcnow().timestamp())
13        time_elapsed = datetime.datetime.utcnow().timestamp() - start_time
14        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
15        return remove_fully_correct_function
16
17    return inside_function
18
19 @elapsed_time
20● def remove_fully_correct(list1, list2):
21
22●     """
23     Returns a list that has all the characters in list1
24     except for those that are in the same position as the
25     same character in list2.
26
27     >>> remove_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
28     ['R', 'O', 'P']
29     >>> remove_fully_correct(['A', 'B', 'O', 'D'], ['D', 'B', 'A', 'D'])
30     ['A', 'O']
31     >>> remove_fully_correct(['B', 'B', 'B'], ['A', 'B', 'B'])
32     ['B']
33     """
34     #Creates the new list
35     new_list = []
36     #Goes through each index value in list1
37     for i in range(len(list1)):
38         #Checks if the letter at the index i in list1 is
39         #not the same letter at the index i in list 2
40         if list1[i] != list2[i]:
41             #Adds that letter to new_list
42             new_list.append(list1[i])
43
44     return new_list
45
46 remove_fully_correct_function = remove_fully_correct(['h', 'e', 'l', 'l', 'o'], ['h', 'l', 'l', 'h', 'h'])
47 print("The remove_fully_correct function will output: " + str(remove_fully_correct_function))

```

```

Console × PyUnit
<terminated> dynamicAnalysis_remove_fully_correct.py [/usr/local/bin/python3.9]
remove_fully_correct start time: 2022-03-24 03:23:02.537911
remove_fully_correct end time: 2022-03-24 03:23:02.538139
remove_fully_correct Elapsed time: 00:00:00.000228
The remove_fully_correct function will output: ['e', 'l', 'o']

```

The `remove_fully_correct` function returns a list that has all the characters in `list1` except for those that are in the same position as the same character in `list2`. The two lists that the function is comparing are `list1=“hello”` and `list2=“hllhh”`. The function returns with a list “elo” and runs with an elapsed time of 228 micro seconds.

```

1• import time
2  import datetime
3
4• def elapsed_time(func):
5
6•     def inside_function(*args):
7
8         start_time = time.time()
9         print(func.__name__, 'start time:', datetime.datetime.utcnow().fromtimestamp(start_time))
10        find_fully_correct_function = func(*args)
11        end_time = time.time()
12        print(func.__name__, 'end time:', datetime.datetime.utcnow().fromtimestamp(end_time))
13        time_elapsed = datetime.datetime.utcnow().fromtimestamp(end_time-start_time)
14        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
15        return find_fully_correct_function
16
17    return inside_function
18
19 @elapsed_time
20• def find_fully_correct(answer, guess):
21
22•     """
23     Returns a list containing the letter "b" for each
24     colour in guess (list of strs) that is in the same position as the
25     same colour in answer (list of strs).
26
27     >>> find_fully_correct(['G', 'B', 'R', 'Y'], ['G', 'R', 'R', 'O'])
28     ['b', 'b']
29     >>> find_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
30     ['b']
31     >>> find_fully_correct(['P', 'G', 'R', 'O', 'Y'], ['Y', 'G', 'O', 'O', 'Y'])
32     ['b', 'b', 'b']
33     """
34     #Creates a list that will store the "b" clues
35     correct_b = []
36     #Goes through each index value in answer
37     for i in range(len(answer)):
38         #Checks if the colour at the index i in guess is
39         #the same colour at the index i in answer
40         if guess[i] == answer[i]:
41             #Adds a "b" to correct_b
42             correct_b.append("b")
43
44     return correct_b
45
46 find_fully_correct_function = find_fully_correct(['h','e','l','l','o'], ['h','h','h','h','h'])
47 print("The find_fully_correct function will output: " + str(find_fully_correct_function))

```

```

[Console X] [PyUnit] [Close]
<terminated> dynamicAnalysis_find_fully_correct.py [/usr/local/bin/python3.9]
find_fully_correct start time: 2022-03-24 03:05:03.134292
find_fully_correct end time: 2022-03-24 03:05:03.134518
find_fully_correct Elapsed time: 00:00:00.000226
The find_fully_correct function will output: ['b']

```

The `find_fully_correct` function returns a list containing the letter "b" for each color in the guess (list of strs) that is in the same position as the same color in the answer (list of strs). The two lists the function is comparing are `answer = "hello"` and `guess = "hhhhh"`. The function returns a list "b" and runs with an elapsed time of 226 micro seconds.

```

1● import time
2● import datetime
3
4● def elapsed_time(func):
5
6●     def inside_fuction(*args):
7
8●         start_time = time.time()
9●         print(func.__name__, 'start time:', datetime.datetime.utcnow().fromtimestamp(start_time))
10●        valid_function = func(*args)
11●        end_time = time.time()
12●        print(func.__name__, 'end time:', datetime.datetime.utcnow().fromtimestamp(end_time))
13●        time_elapsed = datetime.datetime.utcnow().fromtimestamp(end_time-start_time)
14●        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
15●        return valid_function
16
17●     return inside_fuction
18
19 @elapsed_time
20● def valid(user_guess, valid_characters, guess_size):
21
22●     """
23     Returns True if all the characters in user_guess (list) are in
24     valid_characters (str) and the length of user_guess is equal
25     to guess_size (int).
26
27     >>> valid(['H', 'E', 'L', 'L', 'O'], "HELOYT", 5)
28     True
29     >>> valid(['S'], "S", 3)
30     False
31     >>> valid(['G', 'R', 'R', 'Y'], "GRBYOP", 4)
32     True
33     """
34     #Checks if the length of user_guess is equal to guess_size
35     if len(user_guess) == guess_size:
36         #Goes through each character in user_guess
37         for character in user_guess:
38             #Returns False if the character is not found in valid_characters
39             if character not in valid_characters:
40                 return False
41     else:
42         return False
43
44     return True
45
46 valid_function = valid(['h','h','l','l','h'], ['h','e','l','l','o'], 5)
47 print("The valid function will output: " + str(valid_function))

```

```

[Console] X [PyUnit]
<terminated> dynamicAnalysis_valid.py [/usr/local/bin/python3.9]
valid start time: 2022-03-24 03:13:00.528763
valid end time: 2022-03-24 03:13:00.528976
valid Elapsed time: 00:00:00.000213
The valid function will output: True

```

The function valid returns True if all the characters in user_guess (list) are in valid_characters (str) and the length of user_guess is equal to guess_size (int). The function inputs are user_guess = 'hhllh', valid_characters = 'hello', and guess_size = 5. The function returns True and runs with an elapsed time of 213 micro seconds.

```

1@ import time
2 import datetime
3
4@ def elapsed_time(func):
5
6@     def inside_fuection(*args):
7
8         start_time = time.time()
9         print(func.__name__, 'start time:', datetime.datetime.utcnow().fromtimestamp(start_time))
10        find_colour_correct_function = func(*args)
11        end_time = time.time()
12        print(func.__name__, 'end time:', datetime.datetime.utcnow().fromtimestamp(end_time))
13        time_elapsed = datetime.datetime.utcnow().fromtimestamp(end_time-start_time)
14        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
15        return find_colour_correct_function
16
17    return inside_fuection
18
19@ def remove_fully_correct(list1, list2):
20
21@     """
22@     Returns a list that has all the characters in list1
23@     except for those that are in the same position as the
24@     same character in list2.
25
26@     >>> remove_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
27@     ['R', 'O', 'P']
28@     >>> remove_fully_correct(['A', 'B', 'O', 'D'], ['D', 'B', 'A', 'D'])
29@     ['A', 'O']
30@     >>> remove_fully_correct(['B', 'B', 'B'], ['A', 'B', 'B'])
31@     ['B']
32
33#Creates the new list
34new_list = []
35#Goes through each index value in list1
36for i in range(len(list1)):
37@     #Checks if the letter at the index i in list1 is
38@     #not the same letter at the index i in list 2
39@     if list1[i] != list2[i]:
40@         #Adds that letter to new_list
41@         new_list.append(list1[i])
42
43return new_list
44
45 @elapsed_time
46@ def find_colour_correct(answer, guess):
47
48@     """
49@     Returns a list containing the letter "w" for each
50@     character in guess (list of strs) that is also in answer (list of strs)
51@     but isn't in the right position.
52
53@     >>> find_colour_correct(['Y', 'P', 'G', 'G'], ['G', 'P', 'O', 'R'])
54@     ['w']
55@     >>> find_colour_correct(['O', 'P', 'P', 'R'], ['O', 'R', 'P', 'P'])
56@     ['w', 'w']
57@     >>> find_colour_correct(['Y', 'P', 'G'], ['G', 'G', 'O'])
58@     ['w']
59
60#Creates the list that will store all the "w" clues
61correct_w = []
62#Removes all the colours that are in the right position from both answer and guess
63new_guess = remove_fully_correct(guess, answer)
64new_answer = remove_fully_correct(answer, guess)
65#Goes through each colour in new_guess
66for colour in new_guess:
67    #Checks if the colour is in new_answer
68    if colour in new_answer:
69        #Adds a "w" to correct_w

```

```

69         ...     #Adds a "w" to correct_w
70         correct_w.append("w")
71         #Removes that colour from new_answer so that it isn't checked twice
72         new_answer.remove(colour)
73
74     return correct_w
75
76 find_colour_correct_function = find_colour_correct(['y','p','g'], ['g','g','o'])
77 print("The find_colour_correct function will output: " + str(find_colour_correct_function))
78

```

The screenshot shows a Jupyter Notebook interface with a 'Console' tab active. The code in the cell is identical to the one above. The output shows the execution time (start, end, and elapsed time) and the result of the print statement: `The find_colour_correct function will output: ['w']`.

```

<terminated> dynamicAnalysis_find_colour_correct.py [/usr/local/bin/python3.9]
find_colour_correct start time: 2022-03-24 03:23:54.328457
find_colour_correct end time: 2022-03-24 03:23:54.328694
find_colour_correct Elapsed time: 00:00:00.000237
The find_colour_correct function will output: ['w']

```

The function `find_colour_correct` returns a list containing the letter "w" for each character in `guess` (list of strs) that is also in `answer` (list of strs) but isn't in the right position. The function inputs are `answer = 'ypg'` and `guess = 'ggo'`. The function returns '`w`' and runs with an elapsed time of 237 micro seconds.

Challenges Faced:

Now I will go over some of the challenges I faced throughout this assignment:

- The first challenge I faced was learning how to slice a program. I learned about program slicing through the lecture slides and online tutorials. I figured out the types of slicing you can do and started to implement them.
- The next challenge I faced was deciding which level I should apply instrumentation. I decided I would apply instrumentation on the functions. I came to the conclusion that it would be interesting seeing how long each function takes to run.
- Another challenge I was faced with was learning how to apply instrumentation. I solved this problem by watching videos on youtube. I watched a video and learned I can get the elapsed time of a function by creating a function which subtracts the end time with the start time.